下载APP

12｜深度学习（中）：如何用RNN预测激活率走势？

2021-09-24 黄佳

《零基础实战机器学习》 课程介绍 >

讲述：黄佳

时长 22:20 大小 20.46M

你好，我是黄佳。

欢迎来到零基础实战机器学习。在上一讲中，我们通过给鲜花图片分类，学习了 CNN 在图像识别方面的应用。这一讲呢，我们就来学习另外一种深度学习模型——循环神经网络 RNN（Recurrent Neural Network）。那么进入正题之前，我先给你讲个段子，让你直观理解一下，循环神经网络和其它神经网络模型有啥不同。

假如我和你开车去商场，然后我说“嘿！你知道吗，昨天老王的夫人生二胎了！”你说：“是吗？这么大年纪，真不容易。对了，上次你说他的项目没上线，对吧，后来那项目到底怎么样了呢？”我回答说：“嗨，那个项目啊，别提了，把他整惨了，三次上线都失败了，现在公司在考虑放弃老王负责的这个项目。嗯，到了，你等我一会儿，我去给老王买个 _____（此处为填空，选项 1：项目；选项 2：玩具）慰问一下他。”

若是这个选择题给循环神经网络之外的其它神经网络做，我们得到的答案可能是“项目”，因为在输入的文本（特征）中，一直是在谈项目，可正确答案显然是“玩具”。也就是说，如果让神经网络实现类似于人脑的语义判别能力，有一个很重要的前提是，**必须从所有过去的句子中保留一些信息，以便能理解整个故事的上下文**。而循环神经网络中的记忆功能恰恰解决了这个“对前文的记忆”功能。

而这种对前文记忆的功能，让循环神经网络特别适合处理自然语言、文本和时间序列这种“新内容依赖于上下文或者历史信息”的数据。

好，明白了这一点，现在我们来看看这节课要解决的问题吧！

定义问题

今天这个项目的业务场景非常简单：易速鲜花 App 从 2019 年上线以来，App 的日激活数稳步增长。运营团队已经拉出两年以来每天的具体激活数字，现在要交给你的任务是，建立起一个能够预测未来激活率走势的机器学习模型。

	A	B
1	日期	App日激活数
2	1/1/2019	419
3	1/2/2019	432
4	1/3/2019	436
5	1/4/2019	439
6	1/5/2019	439
7	1/6/2019	435
8	1/7/2019	423
9	1/8/2019	428
10	1/9/2019	421
11	1/10/2019	418
12	1/11/2019	423
13	1/12/2019	424
14	1/13/2019	431

你可以在这里下载这个数据集，或者就基于这个数据集，在 Kaggle 网站上创建新的 Notebook。代码呢，我把它发布在Kaggle网站，你也可以 Copy & Edit 它到自己的 Notebook。

看到这个数据集，你可能会觉得和之前的数据集不太一样了，它的字段特别少，只有两个。现在，我给你几秒钟思考一下，这个数据集的特征和标签分别是什么？你可能会说，标签比较容易找，因为要预测激活数，所以“激活数”就是标签。

你可能还会进一步分析说，从机器学习的类型来看，这里我们有要预测的标签，所以它是一个监督学习问题，而且，标签是连续性的数值，因此它是一个回归问题。

这些都没错，不过，特征在哪里呢？难道说，“日期”字段就是这个数据集的特征吗？这些日期看起来像是普普通通的编号，它既不是有意义的值（比如分数），也不是分类的类别编码（比如男生 / 女生），难道机器学习模型这么厉害，输入这样的编号，就能够预测出未来的 App 激活数？

而且，如果你回忆一下我们之前见过的那些数据集，就会发现特征和标签之间会呈现出一种相关性。比如说点赞数、转发数多的时候，往往浏览量也多。但是，在这个数据集中，日期和 App 的激活数之间是不存在这种相关性的。所以，日期并不是一个合适的特征。

不过，在这个数据集中，“日期”仍然是非常重要的信息。让我们一起来回顾一下这个问题的目标：本质上，我们是希望从过去的状态中，找到蛛丝马迹，来预测未来的状态。而过去的状态，肯定是和日期相关联的。

所以，我们可以这样来构建这个数据集的特征：从当前要预测的日期开始，假如是 2020 年 6 月 30 日，我们回推 60 天，拥有从 2020 年 5 月 1 日到 2020 年 6 月 29 日这 60 天的激活数历史数值。那么，如果我们把这些历史数值都做为特征，输入机器学习。并且，我们有理由相信，2020 年 6 月 30 日往后的激活数与前 60 天的激活数有很强的相关性。根据这样的特征数据，机器就可以学习到一条预测激活数的趋势线了。

其实，像这样和日期、时间相关的，并且以连续性数值为特征的数据集，就叫做**时间序列数据**，简称**时序数据**。典型的时序数据包括股票价格、点击数、产品销售数量等等。

那到这里呢，我们就已经把问题定义好了，数据也有了，紧接着下一步就是数据的预处理。

收集数据和预处理

1. 数据可视化

我们先把数据集导入程序。其中 parse_dates=[‘Date’]是把 Date 字段以日期格式导入。

```
import numpy as np #导入NumPy
import pandas as pd #导入Pandas
df_app = pd.read_csv('app.csv', index_col='Date', parse_dates=['Date']) #导入数
df_app #显示数据
```

复制代码

输出如下：

	Date	Activation
0	1/1/2019	419
1	1/2/2019	432
2	1/3/2019	436
3	1/4/2019	439
4	1/5/2019	439
...	...	...
751	1/21/2021	916
752	1/22/2021	925
753	1/23/2021	926
754	1/24/2021	920
755	1/25/2021	932

756 rows × 2 columns

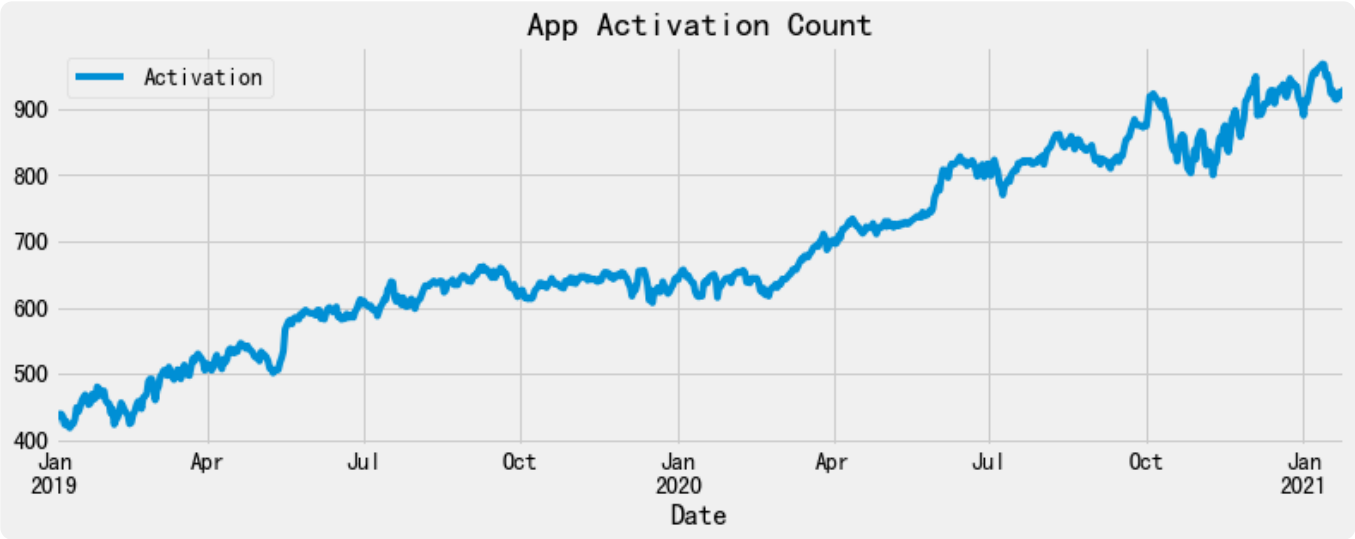
对于时间序列数据，可视化是相当有必要的。我们用 plot 这个 API，绘制出激活数的历史走势图，其中时间为横坐标，App 激活数为纵坐标的图像：

```
import matplotlib.pyplot as plt #导入matplotlib.pyplot
plt.style.use('fivethirtyeight') #设定绘图风格
df_app["Activation"].plot(figsize=(12,4),legend=True) #绘制激活数
plt.title('App Activation Count') #图题
plt.show() #绘图
```

复制代码

这里需要注意的是，我通过语句 plt.style.use(‘fivethirtyeight’) 设定了图像的风格。这种风格是从538 网站（一个对各种事件和趋势进行预测的网站）的画风中移植过来的，很适合展示时序数据。

输出如下：



看得出，自 APP 上线以来，日激活数整体呈上升的走势。从函数图像上看，也没有任何缺失的数据点。下面我们再来看看这个数据集是否需要做数据清洗工作。

2. 数据清洗

我们可以用下面这个语句看看有没有 NaN 值：

```
1 df_app.isna().sum() #有NaN吗？
```

复制代码

结果显示数据集中没有 NaN 值。

对于 App 激活数，我只想保证数据集里全部都是正值，所以，可以用下面的语句查看有没有负值和零值：

```
1 (df_app.Activation < 0).values.any() #有负值吗？
```

复制代码

其中，values.any() 这个 API 表示，只要 Orders 字段中有任意一个值是负值或者零值，就会返回 True。

输出如下：

复制代码

```
False
```

说明整个数据集没有一个 0 值或者负值。那我们也不需要做任何的清洗工作了。接下来，我们进入直接训练集和测试集的拆分。

3. 拆分训练集和测试集

我们假设以 2020 年 10 月 1 日为界，只给模型读入 2020 年 10 月 1 日之前的数据，之后的数据留作对模型的测试。那么，我们就以 2020 年 10 月 1 日为界，来拆分训练集和测试集：

复制代码

```
# 按照2020年10月1日为界拆分数据集
Train = df_app['2020-09-30'].iloc[:,0:1].values #训练集
Test = df_app['2020-10-01':].iloc[:,0:1].values #测试集
```

对于这段代码，我解释几点：

> df_app[:‘2020-09-30’]代表 10 月 1 日之前的数据，用于训练模型；
> df_app[‘2020-10-01’:]代表 10 月 1 日之后，用于对训练好的模型进行测试。
>
> 代码中的 iloc 属性，是 Pandas 中对 DataFrame 对象以行和列位置为索引抽取数据。其中的第一个参数代表行索引，指定“:”就表示抽取所有行；而第二个参数中的“0:1”代表要抽取的列索引的位置为第 2 列，也就是“激活数”这一列。
>
> 最后，.values 这个属性就把 Pandas 对象转换成了 Numpy 数组，我们在上一讲中说过，机器学习中也把 Numpy 数组称为张量，神经网络模型需要 Numpy 张量类型作为输入。

这时候，如果我们显示这个数据对象 Train，就会发现它已经成为了一个 NumPy 数组：

复制代码

```
Train #显示训练集对象
```

输出如下：

复制代码

```
array([[419],
       [432],
       ...
       [872],
       [875]])
```

对于神经网络来说，输入的张量形状非常重要，我们要显式指明输入维度，绝不能错，所以我们要经常用 NumPy 中的.shape 属性查看当前数据对象的形状。

复制代码

```
print('训练集的形状是：', Train.shape)
print('测试集的形状是：', Test.shape)
```

输出如下：

复制代码

```
训练集的形状是： (639, 1)
测试集的形状是： (117, 1)
```

目前，训练集是 639 行的一维数组，测试集是 117 行的一维数组，但它们都是二阶张量。

为了直观地显示拆分，我们可以把原始数据集按照拆分日期用不同颜色进行显示。

复制代码

```
# 以不同颜色为训练集和测试集绘图
df_app["Activation"][:'2020-09-30'].plot(figsize=(12,4),legend=True) #训练集
df_app["Activation"]['2020-10-01':].plot(figsize=(12,4),legend=True) #测试集
plt.legend(['Training set (Before October 2020)','Test set (2020 October and b
plt.title('App Activation Count') #图题
plt.show() #绘图
```

输出如下：

好啦，拆分出训练集和测试集后，下面我们做特征工程。

4. 特征工程

对于神经网络来说，这是一个特别重要的步骤。我们上节课说过，神经网络非常不喜欢数值跨度大的数据，所以，我们对训练特征数据集进行归一化缩放。

复制代码

```
from sklearn.preprocessing import MinMaxScaler #导入归一化缩放器
Scaler = MinMaxScaler(feature_range=(0,1)) #创建缩放器
Train = Scaler.fit_transform(Train) #拟合缩放器并对训练集进行归一化
```

归一化完成之后，我们来做最后一个数据预处理的步骤：构建特征集和标签集。

5. 构建特征集和标签集

我们前面说过，这个数据集的标签就是 App 激活数，特征是时序数据。如果我们要预测今天的 App 下载数量，那时序数据特征的典型构造方法就是，把之前 30 天或者 60 天的 App 下载数量作为特征信息被输入机器学习模型。

所以，在下面的这段代码中，我们创建了一个具有 60 个时间步长（所谓步长，就是时间相关的历史特征数据点）和 1 个输出的数据结构。对于训练集的每一行，我们有 60 个之前的 App 下载数作为特征，1 个当前训练集元素作为标签。

复制代码

```
# 创建具有 60 个时间步长和 1 个输出的数据结构 - 训练集
```

```
X_train = [] #初始化
y_train = [] #初始化
for i in range(60,Train.size):
    X_train.append(Train[i-60:i,0]) #构建特征
    y_train.append(Train[i,0]) #构建标签
X_train, y_train = np.array(X_train), np.array(y_train) #转换为NumPy数组
X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1)) #转换成神经
```

在这个过程中的最后一步，我是用 NumPy 中的 reshape 方法，把特征数据集转换成神经网络所需要的形状的。我之所以强调这一点，是因为如果形状不对，程序在训练时就会报错，这个错误对初学者来说很常见。

这里我显示一下目前 X_train 集的形状：

复制代码

```
X_train.shape #X_train的形状
```

输出如下：

复制代码

```
X_train的形状是： (579, 60, 1)
```

输出显示它是一个三阶张量，每一阶上面的维度是 579（579 行数据）、60（每一个当日激活数往前 60 天的激活数）和 1（每一个时间点只有激活数一个特征）。

我们用同样的方法构建测试集：

复制代码

```
TrainTest = df_app["Activation"][:] #整体数据
inputs = TrainTest[len(TrainTest)-len(Test) - 60:].values #Test加上前60个时间步
inputs = inputs.reshape(-1,1) #转换形状
inputs  = Scaler.transform(inputs) #归一化
# 创建具有 60 个时间步长和 1 个输出的数据结构 - 测试集
X_test = [] #初始化
y_test = [] #初始化
for i in range(60,inputs.size):
    X_test.append(inputs[i-60:i,0]) #构建特征
    y_test.append(inputs[i,0]) #构建标签
```

```
X_test = np.array(X_test) #转换为NumPy数组
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1)) #转换成神经网络
```

如果我们显示测试集的形状，会发现它是形状为 (117, 60, 1) 的张量。

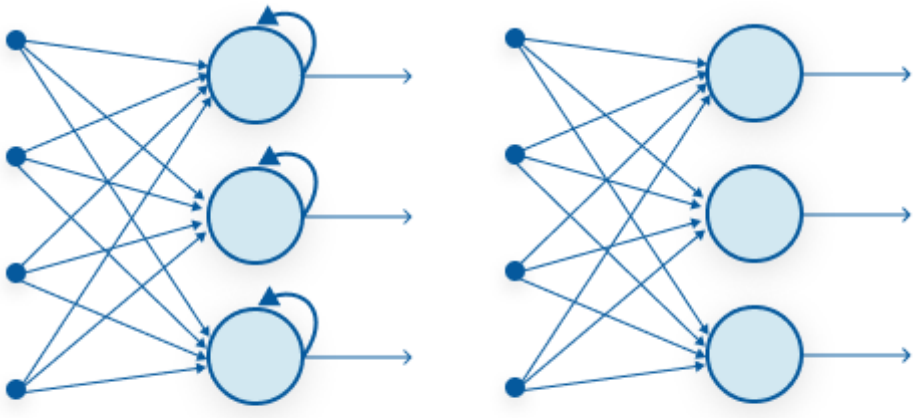
好啦，至此，数据预处理的工作就终于完成了。我们进入到下一个环节：选择算法，创建模型。

选择算法并建立模型

前面我已经给你剧透了，对于预测 App 未来激活数的问题，我们会选择神经网络中的 RNN。其实，我们可选的算法还是挺多的，比如说普通的回归算法，还有像 AR（自回归）、MA（滑动平均）、ARMA（自回归滑动平均）、ARIMA（差分自回归滑动平均）等专门处理时序数据的模型，还有 Facebook 发布的 Prophect 算法，也可以用于时间序列的预测。

当然，最为高效、最常用的时序预测模型，仍然莫过于深度学习中的 RNN 了。这也是我们这次实战中会选用 RNN 模型。那么，为什么 RNN 擅长时序数据的处理呢？这里我们就要多说两句它的原理了。

我们知道，循环神经网络是神经网络的一种，不过，和其它类型的神经网络相比，最大的不同是它建立了自身的记忆机制，增加了时间相关的状态信息在各层神经元间的循环传递机制。即一个序列的当前输出与前面各个神经元的输出也是有关的，即隐藏层之间不再是不相连的，而是有连接的。这就让循环神经网络能够更加自由和动态地获取输入的信息，而不受到定长输入空间的限制。



循环神经网络建立了与时间状态相关的记忆机制

因此，向循环神经网络中输入的数据都有这样的特点， 数据集中不仅要包含当前的特征值，还要包括前一刻或者前几刻的状态。

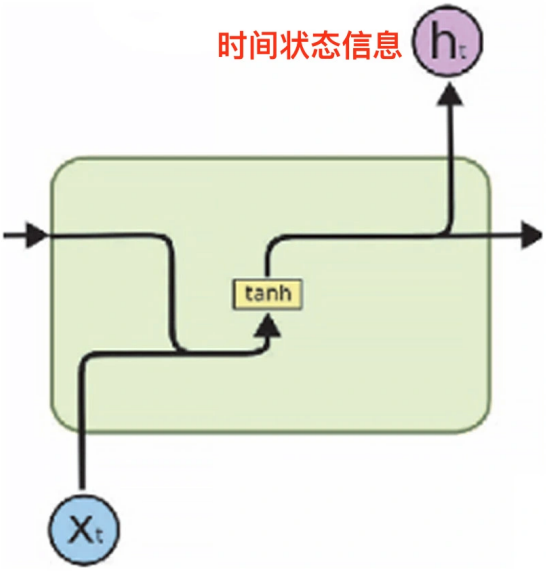
循环神经网络的应用场景正是这种有“上下文历史”的数据，比如说我们文章开头说的那个给老王买礼物的例子，那其实是在对自然语言处理。再比如视频处理，当前帧中发生的事情在很大程度上取决于之前数帧中的内容。另外，还有一个典型的应用场景就是我们这里的实战案例，循环神经网络要处理的是一个典型的时间序列分析数据集，根据前面 60 天的激活数，对未来的数字进行预测。

你会发现，这些场景和数据集，都有一个显著的特点，就是要预测的内容，和之前一段时间的特征值密切相关，就适合选择循环神经网络来建立模型进行处理。

到这里，你应该理解了为什么我们的时序数据预测要用 RNN 来处理，现在，我们就来考虑怎么搭建 RNN。

在 Kares 中，主要有三种循环神经网络层可以搭建循环神经网络，分别是 Simple RNN、LSTM 和 GRU，那我们该选哪一个呢？我们来做个甄别。

Simple RNN，顾名思义，就是最简单的循环神经网络结构，它的结构如下图所示。这个结构比较简单，只是在输入特征 X 的基础之上加入了 h_t 这个时间状态信息，也就是“记忆”功能。



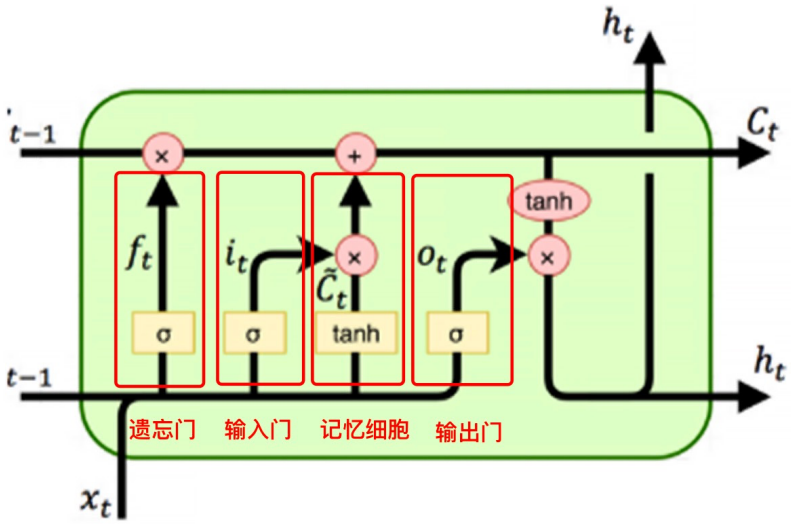
Simple RNN神经元结构

不过这种结构有一个缺陷，就是会出现“短期记忆的问题”。我们知道，神经网络在训练的过程中，参数是从后面的层向前面的层反向传播的，同时还会在每一步计算梯度，做梯度下降，用于更新神经网络中的权重。

如果前一层对当前层的影响很小，那么梯度值就会很小，反之亦然。如果前一层的梯度很小，那么当前层的梯度会更小。这就使得梯度在我们反向传播时呈指数缩小。而较小的梯度意味着它不会影响权重更新。所以，对于 Simple RNN 来说，较早期输入的信息对预测判断的影响会比较小，这就是我们前面说的“短期记忆问题”。

对于我们这个项目的数据集来说，时间跨度比较大，Simple RNN 是很难捕捉到这种长期的时间关联的。

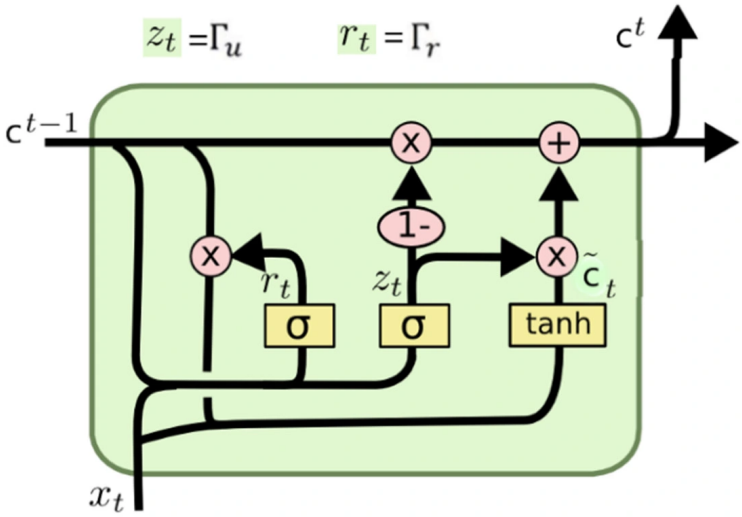
不过，LSTM（Long Short-Term Memory，长短期记忆网络）可以很好地解决这个问题。LSTM 的神经元由一个遗忘门、一个输入门、一个输出门和一个记忆细胞组成，来记录额外的信息。记忆细胞负责记住时间相关的信息，而三个门负责调节进出神经元的信息流。这个过程中的数学原理我们这里不详述，你只需了解在这个过程中，每个记忆单元可获得连续的梯度流，能学习数百个时间步长的序列而误差保持原值，从而解决梯度消失问题。



LSTM神经元结构

所以说，LSTM 网络可以弥补 Simple RNN 对较长时期前历史信息相对不敏感的缺陷，它也被称为“穿越时空的旅程”，它的这种结构非常适合处理时间序列数据。

那么 GRU 呢？它适不适合我们这个项目呢？其实，GRU 也是为了解决 Simple RNN 的短期记忆问题，它的复杂性介于 Simple RNN 和 LSTM 之间，在结构上要比 LSTM 简单一些。这里你只需要理解 GRU 是速度和性能的折衷选择就行。



GRU神经元结构

对于预测 App 激活数走势这个项目来说，如果仅从性能角度考虑，那 LSTM 是最理想的，所以，我们就构建一个以 LSTM 为主要层结构的循环神经网络：

复制代码

```
from tensorflow.keras.models import Sequential #导入序贯模型
from tensorflow.keras.layers import Dense, LSTM #导入全连接层和LSTM层
# LSTM网络架构
RNN_LSTM = Sequential() #序贯模型
RNN_LSTM.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[
RNN_LSTM.add(LSTM(units=50, return_sequences=True)) #中间1层LSTM，return_sequen
RNN_LSTM.add(LSTM(units=50, return_sequences=True)) #中间2层LSTM，return_sequen
RNN_LSTM.add(LSTM(units=50)) #中间3层LSTM
RNN_LSTM.add(Dense(units=1)) #输出层Dense
# 编译网络
RNN_LSTM.compile(optimizer='rmsprop', #优化器
                 loss='mean_squared_error') #损失函数
RNN_LSTM.summary() #输出神经网络结构信息
```

这个神经网络的结构如下所示：

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 60, 50)            10400
_________________________________________________________________
dropout (Dropout)            (None, 60, 50)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 60, 50)            20200
_________________________________________________________________
dropout_1 (Dropout)          (None, 60, 50)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 60, 50)            20200
_________________________________________________________________
dropout_2 (Dropout)          (None, 60, 50)            0
_________________________________________________________________
lstm_3 (LSTM)                (None, 50)                20200
_________________________________________________________________
dropout_3 (Dropout)          (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 71,051
Trainable params: 71,051
Non-trainable params: 0
_________________________________________________________________
```

到这里呢，我们就搭建好了合适的循环神经网络模型，搭建的过程和上一讲中搭建 CNN 的方法非常相似。接下来，我们开始做模型训练。

训练并评估模型

这里我们训练 50 次，并在训练的同时进行 80/20 比例的数据验证：

复制代码

```
history = regressor.fit(X_train, y_train, epochs=50, validation_split=0.2) # 训
```

输出如下：

复制代码

```
Epoch 1/50
15/15 [==============================] - 11s 258ms/step - loss: 0.0813 - val_l
Epoch 2/50
15/15 [==============================] - 2s 122ms/step - loss: 0.0101 - val_lo
Epoch 3/50
15/15 [==============================] - 2s 117ms/step - loss: 0.0125 - val_lo
... ....
Epoch 48/50
```

```
15/15 [==============================] - 2s 118ms/step - loss: 0.0056 - val_lo
Epoch 49/50
15/15 [==============================] - 2s 121ms/step - loss: 0.0033 - val_lo
Epoch 50/50
15/15 [==============================] - 2s 119ms/step - loss: 0.0036 - val_lo
```

由于这是一个回归值的预测问题，没有分类准确率指标，仅有损失值这个指标，我们就用上一讲中说过的损失曲线来显示训练过程中损失值的变化：

可以看到，训练 50 轮之后，训练集上的损失已经很小了。不过，测试集上面的损失存在着振荡上升现象，这也是过拟合的一个标志。

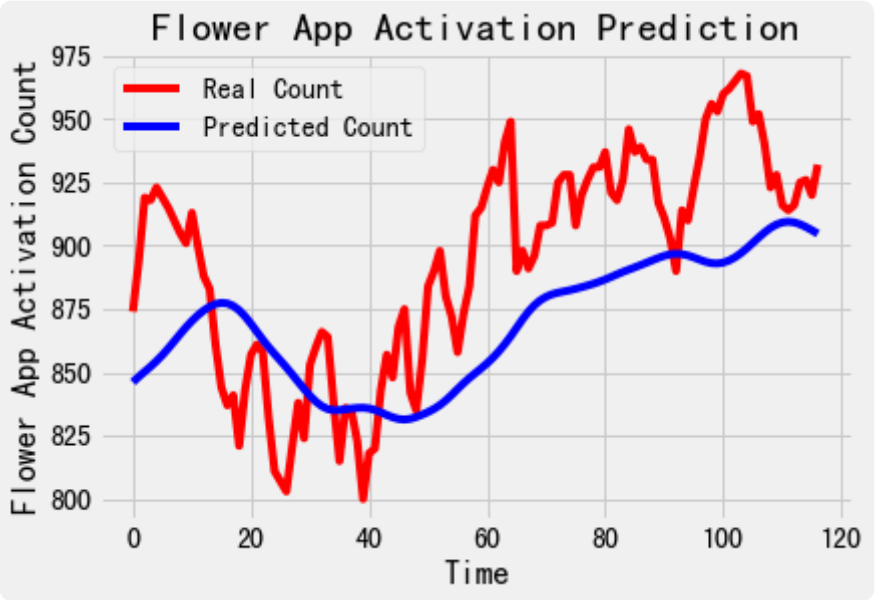
利用模型进行预测

好啦，现在模型有了，评估也结束了，现在我们就来到了最令人激动的环节，让我们用模型预测一下测试集（也就是代表着未来）的激活数吧。不过，在预测结束之后，需要用 inverse_transform 对预测值做反归一化。否则，激活数将是一个 0-1 之间的值。

复制代码

```
predicted_stock_price = regressor.predict(X_test) #预测
predicted_stock_price = sc.inverse_transform(predicted_stock_price) #反归一化
plot_predictions(test_set,predicted_stock_price) #绘图
```

输出如下：



看得出来，我们的回归曲线和实际曲线的走势是相当接近的。不过，这个模型是否有进一步优化的可能，这个问题我们下一讲继续探讨。那到这里，我们的 App 激活率走势预测的任务就顺利结束了。

总结一下

现在，我们来回顾一下重点内容。

在这一讲中， 我们介绍了时间序列数据，以及善于处理时间序列数据集的循环神经网络算法 RNN。RNN 和其它类型的神经网络相比，它的特点是建立了自身的记忆机制，善于根据历史信息预测后续走势。通过 Keras 中的 API，我们轻松搭建起了一个循环神经网络对数据集，进行 App 激活数的预测，而这个神经网络中最主要的层是 LSTM 层。

还有一点请你注意，要通过历史数据预测未来是非常有难度的事情，所以，我们并不能够单纯的依赖于模型给出的结果。在实际情况中，影响未来的因子有很多，就拿股票的价格为例，一个预测系统不仅仅要有历史股价，还要结合当前市场的整体行情、新闻信息、公司财报等多方面的数据来建立模型才较为完善。但即使如此，我们仍然是很难预测走势的。像我们的 App 激活数的预测，会比股票价格预测简单一些，那也需要更多的产品、运营等更多信息，把这些信息和历史数据同时传递给模型，才能够训练出更有意义的模型。

思考题

这节课就到这里了，我给你留一个思考题：

我们说，在 Keras 中，一共有三种主要的循环神经网络层，分别是 SimpleRNN、GRU 和 LSTM，我们今天使用了其中最强大的 LSTM 层，请你试着用其它两种解决今天的问题。

欢迎你在留言区和我分享你的观点，如果你认为这节课的内容有收获，也欢迎把它分享给你的朋友，我们下一讲再见！

极客时间

本讲重点

1. 循环神经网络的神经元间存在循环传递机制，这使得它存在记忆功能，适合处理自然语言、视频和时间序列信息。

2.Keras 提供三种循环神经网络层，分别是 SimpleRNN、GRU 和 LSTM。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

赞 0 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 11 | 深度学习（上）：用CNN带你认识深度学习

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。